一种基于循环编码的高性能分布式互斥算法

李美安, 刘心松, 王 征
(电子科技大学计算机学院 8010 研究室, 四川成都 610054)

摘 要: 公平、健壮和易于实现的分布式互斥算法对分布式系统保证数据一致性、逻辑一致性及时序一致性至关重要. 除 Lamport 算法, RA 算法和 $N^{0.63}$ 算法外, 以前提出的分布式互斥算法都只是在节点数目与请求集大小存在一定关系时才是公平和对称的, 在大多数情况下是不对称的. 这些算法的同步时间, 容错性能与消息复杂度之间存在着不可调和的矛盾, 不能三者兼顾. 本文提出了一种基于循环编码的互斥请求集产生算法, 并在此基础上改进了已有的基于请求集的分布式互斥算法, 使该算法在系统节点数为任意值时都能公平和对称地产生请求集. 其消息复杂度较低, 同步时间为 T, 节点容错能力达到 $N-1$.

关键词: 循环编码; 分布式; 互斥
中图分类号: TP393 文献标识码: A 文章编号: 0372-2112 (2005) 08-1397-06

A High Performance Distributed Mutual Exclusion Algorithm Based on Cyclic Coding

LI Mei-an, LIU Xin-song, WANG Zheng
(8010 *R&D Group, University of Electronic Science and Technology of China, Chengdu, Sichuan* 610054, *China*)

Abstract: It's very important to use a fair and easy implementation distributed mutual exclusion algorithm to ensure the data, logic and time consistency of a distributed system. Except the Lamport, RA and $N^{0.63}$ algorithm, all the other distributed mutual exclusion algorithms brought forward formerly are fair only when the number of the system sites is peculiarly related on the sites number of the quorum. It's not fair in most instances. And there are some contradiction in synchronization time, message complexity and fault tolerance. A new distributed mutual exclusion algorithm has been proposed that is based on cyclic coding in this paper. This algorithm can reduce the message complexity to $O(2\sqrt{N})$ and is fair to any scale distributed system. And the synchronization time has been reduced to T and the fault sites tolerance can be $N-1$.

Key words: cyclic coding; distributed; mutual exclusion

1 引言

互斥是实现计算机系统临界资源访问控制的重要手段. 在分布式系统中, 互斥作为保证多副本对象内容一致性和逻辑一致性的基本技术, 其算法控制及实现都较单机系统复杂. 在过去近 40 年的时间里对分布式互斥算法的研究取得了很大进展. 对全分布互斥算法做出里程碑贡献的有 Lamport[1], Richard& Agrawala[2] 和 Maekawa[3]. Lamport 提出了逻辑时戳(Logical Timestamp) 的概念. 对没有全局物理时钟的分布式系统, 消息可以根据逻辑时戳表示的优先级进行全局排序. Richard&Agrawala 巧妙地利用延迟应答的方式将系统中各进程的请求按逻辑时戳组成一个动态令牌环. 他们还将应答与释放消息合而为一, 将算法的一次请求所需消息数降为 $2(N-1)$. Maekawa 提出了请求集(Quorum) 的概念, 并利用有限射影平面理论得出了基于竞争允许的分布式互斥一次请求消息数的下限. 将算法的消息复杂度降为 $O(\sqrt{N})$.

在以上三种算法的基础上, 研究人员提出了各种优化算法. 有基于消息复杂性优化的算法如文[4~12]. 有基于可靠性优化的算法如文[13, 14], 还有基于同步时间优化的算法如文[15, 16]. 各种优化算法中, 基于请求集的消息优化算法是目前研究得最深入, 也是最成熟的算法.

Maekawa 在文献[3]中, 介绍了两种请求集的产生算法. 但第一种算法需要利用 $N=m(m-1)+1$ 条件下求得的请求集, 将最后 $m(m-1)+1-N$ 个节点用第 $2N-((m(m-1)+1)$ 到第 $m(m-1)+1-N$ 个节点进行置换才能得到. 由于节点 $2N-((m(m-1)+1)$ 到 $m(m-1)+1-N$ 被重复置换, 该算法不能满足文献[3]中提出的请求集元素相等的条件, 也不能满足任一个节点属于相同数量请求集的条件, 因此为不平衡算法. 第二个算法在 $N=m(m-1)+1$ 条件下满足文献[3]中提出的 4 个条件, 但由于每个请求集有两个相交的节点,

收稿日期: 2004-11-08; 修回日期: 2005-01-30

请求集大小变为 2 $\sqrt{N}$, 算法的消息复杂度为 $O(2\sqrt{N})$. 同时, 在不满足 $N = m(m-1)+1$ 条件时, 该算法也是非平衡算法. Luk 在文献[13]中提出了基于三角形逻辑拓扑结构的请求集产生算法, 除位于三角形边上的节点外, 其它节点的请求集是对称的, 且它将算法的消息复杂度降低为 $O(\sqrt{2}\sqrt{N})$. 文献[17]还提出了一种循环的请求集产生算法, 将互斥的消息复杂度降到了接近 $O(\sqrt{N})$ 的水平, 但该算法只是提出并证明了基于循环对称区组设计产生对称请求集的可能性, 并列出了 4 到 111 的节点系统的循环基, 却没有明确提出请求集的产生算法, 对超出 111 个节点的系统, 需要重新验证循环基存在的条件. 因此产生请求集的计算量很大, 无法得到广泛应用. WEE 在文献[18]中提出一种不限制请求集相交节点数的分布式互斥算法, 将互斥的消息复杂度降到了 $O(N^{0.63})$. 但它也具有与文献[17]的循环算法类似的问题, 同样不易推广.

本文通过循环编码产生请求集的方式, 得出了一种消息复杂度较低, 容错性能较高, 且同步时间短的对称分布式互斥算法, 其请求集的产生算法十分简单, 且容易实现. 虽然计算量较大, 但理论上它可以处理任意多个节点的分布式系统互斥访问的请求集产生问题.

2 系统模型

设系统的节点数为 N, 并从 0 到 $N-1$ 对节点编号, 第 i 个节点的 ID 号为 $i-1$. 假定系统的节点与通信均可靠. 各节点没有共享存储器和共同的物理时钟, 节点间依靠消息进行通信. 消息通信有延迟但延迟时间无法预知.

假定系统中各节点都能够及时感知系统状态. 因此, 各节点对系统状态的记录能够在一定时间内逐渐趋于并达到一致.

3 请求集产生的算法思想

基于逻辑拓扑结构的算法, 能够从逻辑拓扑结构上通过观察找出与本节点相关的请求集, 其算法思想简单明了. 如基于网格结构的请求集产生算法, 只须将节点所在行与所在列的节点归于本节点的请求集即可. 同时可将包含本节点的其它请求集作为备选请求集以提高节点容错能力. 基于逻辑拓扑结构的算法由于请求集的产生依赖于拓扑图形, 因此缺乏灵活性, 不能有效降低算法的消息复杂度. 因此有些文献如[17, 18]采用类似二元编码的方式表示节点的请求集, 并将区组设计的理论用于请求集的产生. 但由于它们都采用代数理论进行计算, 又缺乏基于逻辑图形请求集产生算法的直观性. 其灵活性也不能令人满意. 因此, 将二者有效结合, 提出新的请求集产生算法是很必要的.

对节点数为 N 的分布式系统, 用一个长度为 N 的码字表示某个节点的请求集, 码字中位置 i 的数值代表该请求集中系统 ID 号为 i 的节点是否被占用. 1 表示被占用, 0 表示不被占用. 例如, 码字 101100001000 是 12 个节点的系统的一个请求集码字, 它表示该请求集中包含 1, 3, 4, 9 共 4 个节点. 基于循环编码的请求集产生算法要求系统中任意节点的请求集码字能够由其中一个请求集码字循环移位产生. 如节点 i 的请求集码字为 101100001000, 则节点 $i+1$ 的码字为 010110000100. 因此, 系统中任何节点的请求集码字都能够由第 0 个节点的请求集码字循环移位产生. 称系统第 0 个节点请求集码字为系统请求集的循环基. 码字中 1 的个数称为码字的重量. 循环码中各节点对应请求集码字的重量都相同. 码字的重量表示请求集中包含节点的个数. 将系统所有节点的请求集码字按节点 ID 号排在一起构成一个 N 阶方阵, 称它为系统请求集矩阵.

Maekawa 在文献[3]中, 提出了请求集应满足的 4 个条件. 即(1)任意两个请求集的交集不为空. 对用循环码表示的请求集, 该条件可表述为任意一个循环码, 其循环移位后产生的码字与原码字的模 2 加所得码字的重量小于原码字总量的 2 倍. (2)任意节点的请求集必包含该节点本身, 这一条件是为了减少进程每次访问临界区所需的消息数. 对任意节点 i 用循环码表示的请求集, 其码字的第 i 位必为 1. (3)每个请求集的大小相同. 循环码表示的请求集, 由于每个码字的重量都相同, 因此该条件自动满足. (4)任一个节点都属于 n 个请求集, 其中 n 是节点请求集所包含的元素个数. 对用循环码表示的节点请求集, 该条件要求系统请求集矩阵的每一行每一列含有数量相同的 1.

循环码自动满足上述 4 个条件中的(3). 由于系统任意节点的请求集码字能够由其它节点的请求集码字循环移位产生, 例如假定第 0 个节点的请求集码字的第 0 位为 1, 能够满足上述条件(2), 则第 1 个节点的请求集码字能够由第 0 个节点的码字循环产生, 当该码字由第 0 个节点的请求集码字向右循环移动一位产生, 则产生的码字第 1 位为 1. 即第 1 个节点的请求集码字能够满足上述条件(2). 因此, 如果系统第 0 个节点的请求集码字能够满足上述条件(2), 则只要第 i 个节点的请求集码字由第 0 个节点的请求集码字向右循环移动 i 位产生, 则第 i 个节点的请求集码字就能够满足上述条件(2).

由于第 0 个节点的请求集码字中含有 n 个 1, 每个 1 与码字中第 0 位的距离不同. 这些距离大于等于 1, 小于等于 $N-1$. 而码字最后一位向右循环移动一位能够移到第 0 位的位置. 因此, 按循环右移产生的系统的请求集矩阵中, 包含了第 0 行码字向右移动 1 到 $N-1$ 位的各种码字. 系统请求集矩阵的第一行的各个 1 循环向右移动不同距离(1 到 $N-1$)后都会到达矩阵的第一列, 因此, 矩阵的第一列包含矩阵第一行相同数量的 1. 同理, 矩阵的其它列也包含与第 1 列相同数量的 1. 因此, 循环右移产生的系统请求集矩阵满足上述条件(4).

要满足上述条件(1), 必须要求系统请求集矩阵的任意两行至少有一个位置同为 1. 由于系统请求集矩阵的第 i 行由第 0 行向右循环移动 i 位后得到, 同时, 第 i 行也可由第 1 行向右循环移动 $i-1$ 位后得到, 或者由第 k 行($1 \leqslant k < i$)向右循环移动 $i-k$ 位得到. 因此, 满足上述条件(1), 只需让系统请求集矩阵的第 i 行($1 \leqslant i < N-1$)与第 0 行有共同位置为 1 即可.

上述分析揭示了一种系统请求集矩阵的构造方法, 即(1)请求集矩阵的第 i 行($1 \leqslant i < N-1$)与第 0 行有共同位置

为*1*, 因此第 1 行与第 0 行有共同位置为 1. 假定系统请求集矩阵的初始状态为其所有元素均为未知. 由于第 0 行的第 0 位为 1 才能满足上述条件*(2)*, 因此, 可以设请求集矩阵的第 0 行第 0 位为 1. 又循环右移能够满足上述条件*(3)(4)*, 因此, 第 1 行应为第 0 行循环向右移动一位所得, 所以第 1 行的第 1 位为 1. 第 0 行与第 1 行相同位置为 1, 可以设定第 1 行的第 0 个位置为 1 或第一行的第 1 个位置为 1. 如果设定第 1 行的第 0 个位置为 1, 则表示设第 0 行的第 $N-1$ 个位置为 1. 设第 0 行的第 1 个位置为 1, 也表示设第 1 行的第 2 个位置为 1. 如此类推. 当第 i 行与第 0 行已经有相同位置为 1 后, 如果第 $i+1$ 行与第 0 行没有共同位置元素为 1, 则找出第 $i+1$ 行中已知元素的最大位置, 设第 0 行的对应位置元素为 1, 从而保证第 $i+1$ 行与第 0 行有共同位置元素为 1. 设定第 0 行的相应元素后, 标记各行与第 0 行的相交情况, 找出下一个与第 0 行没有共同位置为 1 的行, 重复上述步骤直到所有的行与第 0 行都有共同位置为 1 停止.

4　请求集产生算法的描述与实现

4.1　数据结构

系统请求集矩阵 $\boldsymbol{A}_N$. 它是 $N*N$ 矩阵. 其行列编号均为从 0 到 $N-1$. $\boldsymbol{A}_N$ 的第 i 行表示系统的第 i 个节点的请求集码字, 用 a_{ij} 表示 $\boldsymbol{A}_N$ 的第 i 行第 j 列元素, 它的值表示节点 j 在第 i 个节点的请求集码字中是否被选中. a_{ij} 为 1 表示选中, 为 0 表示不被选中.

标记数组 $\boldsymbol{T}_N$. 它具有 N 个分量, 每个分量对应 $\boldsymbol{A}_N$ 的一行. 该分量为 1 表示 $\boldsymbol{A}_N$ 的对应行与 $\boldsymbol{A}_N$ 的第 0 行有共同位置为 1*(*即两个码字集合的交集不为空*)*. 分量为 0 表示 $\boldsymbol{A}_N$ 的对应行与 $\boldsymbol{A}_N$ 的第 0 行没有共同位置为 1*(* 即两个码字集合的交集为空*)*. 用 t_i 表示 T_N 的第 i 个分量.

4.2　请求集产生算法的描述

(1) 初始化. 令 $\boldsymbol{A}_N=0$, $\boldsymbol{T}_N=0$;

(2) 令 $a_{00}=1$, 循环右移第 0 行产生新的第 1 到第 $N-1$ 行, 置 $t_0=1$;

(3) 找出 $t_i=0$ 的最小的 i, 找出 $a_{ij}=1$ 但 $a_{0j}=0$ 的最大的 j, 令 $a_{0j}=1$, 循环右移第 0 行产生新的第 1 到第 $N-1$ 行, 置 $t_i=1$; 如果还有 $t_i=0$, 转*(3)*, 否则 t_i 全为 1, 算法结束.

4.3　请求集产生算法的实现实例

以 $N=16$ 为例, 图 1 到图 4 描述了各节点请求集的求取过程.

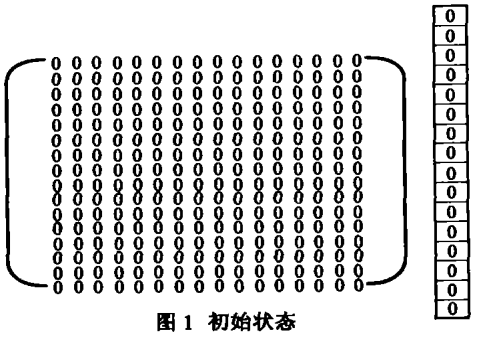

图 1　初始状态

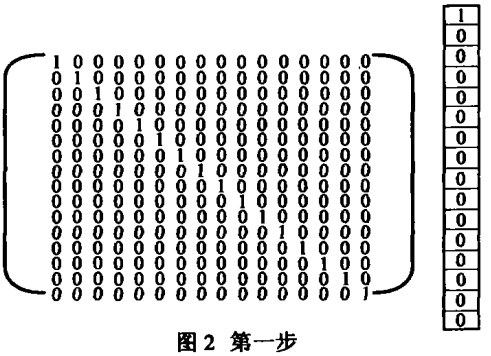

图 2　第一步

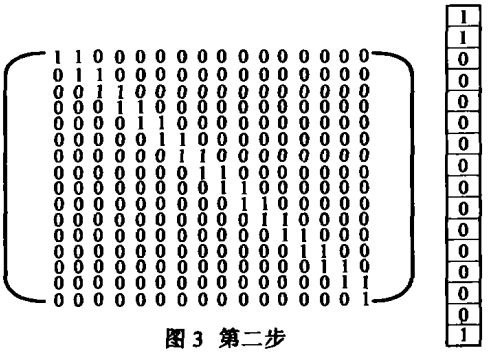

图 3　第二步

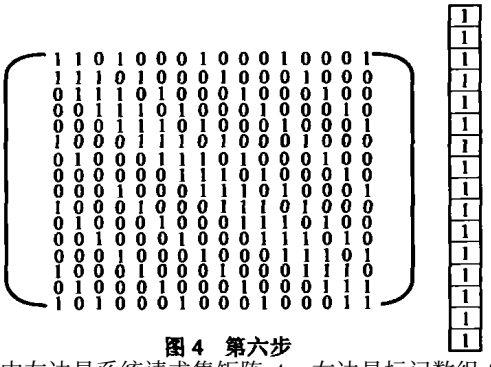

图 4　第六步

图中左边是系统请求集矩阵 $\boldsymbol{A}_N$, 右边是标记数组 $\boldsymbol{T}_N$. 请求集矩阵的第 0 行中 1 的数量反映了求取系统请求集矩阵所需的步骤, 也反映了请求集的长度. 1 的位置表示请求集所包含的节点.

4.4　请求集产生算法优化

同样规模的分布式系统, 其请求集的长度与系统请求集矩阵的初值位置有关. 例如对 16 个节点的系统, 按上述算法得出的请求集循环基是 1101000100010001, 请求集长度是 6. 当令初值为 $a_{03}=1$ 时, 将矩阵的后 4 行插入第 0 行前面构成新的系统请求集矩阵, 则其请求集循环基变为 1010001100000010, 其请求集长度变为 5. 与由 Maekawa 算法求出极限值相同. 因此, 利用上述请求集产生算法, 将计算请求集矩阵的起点从 0 到 $N-1$ 逐一计算请求集长度, 找出产生最小请求集的计算起点 $a_{0j}=1$ 及相应的系统请求集矩阵 $\boldsymbol{A}_{Nj}$, 将 $\boldsymbol{A}_N$ 的后 $j+1$ 行移到 $\boldsymbol{A}_{Nj}$ 的第 0 行前构成新的系统请求集矩阵 $\boldsymbol{A}_N$, $\boldsymbol{A}_N$ 即是基于循环编码产生的最优请求集生成矩阵. 将其作为系统请求集矩阵, 能够保证 Maekawa 提出的请求集需满足的 4 个条件, 同时还能保证基于循环编码产生的

请求集最小.

5 分布式互斥算法描述

本互斥算法与 Makawa 算法基本类似,但为降低同步延迟,提高容错性能和避免死锁,确保算法的健壮性和高效性,本算法采用 Lamport 逻辑时戳作为排队等待的优先级依据,并改变了部分消息的传送方向,增加了转移应答消息和请求集重组过程.

5.1 消息类型

本算法要求四类消息.即请求消息(REQUST),应答消息(REPLY),转移应答消息(TRANS_REP),释放消息(RELEAS).

REQUST 消息和 RELEAS 消息与所有 Makawa 类算法的这两类消息结构与作用都相同. REPLY 消息除包含 Makawa 算法中 REPLY 消息的内容外,如果需要发送 REPLY 消息的请求的后续请求是异地请求,则 REPLY 消息将包含该后续请求的信息.转移应答消息(TRANS_REP)是本算法增加的一类消息.当收到带后续请求的 REPLY 消息时,除完成 Makawa 算法中收到 REPLY 消息的操作,如果收到 REPLY 消息的请求是优先级最高的请求,则将优先级最高的后续请求保存.在释放请求时,向优先级最高的后续请求的发起节点发送 TRANS_REP 消息.TRANS_REP 消息包含发送含有该后续请求的 REPLY 消息的节点.当收到 TRANS_REP 消息时,将 TRANS_REP 消息包含的发送 REPLY 消息的节点将被取出,并将 TRANS_REP 消息当作该节点发送的 REPLY 消息.当系统有部分节点失效,使所有的请求集都不可用时,检测到节点失效的节点将调用请求集重组过程重新生成请求集.

5.2 数据结构

本算法需要各类消息结构,如请求结构(REQ),等待队列结构(RQ),请求集(Quorum),应答集(Rset).请求集是一个数组,包含本节点请求集及后备的请求集,它将根据系统运行情况及时更新.应答集是一个位图,标志是否收到请求集中节点的应答消息.应答集对应位为 0,表示未收到应答,对应位为 1,表示已收到应答.

5.3 算法描述

本算法包含检测请求集,发送请求、释放请求,收到请求,收到应答,收到转移应答,收到释放及请求集重组消息八个过程.并分别描述如下.

(1) 检测请求集 **Check_R_quorum(REQ_i)**

该过程用于检查请求 REQ_i 的应答集 $Rset_i$ 是否已收到所有应答,如果收到所有应答,则 Check_R_quorum(REQ_i)的值为 1,否则为 0.

(2) 发送请求 **REQ_i**

(a) 生成请求 REQ_i,将 REQ_i 按优先级插入等待队列 RQ_i 中.

(b) 读取系统活跃节点并生成 S_i 的请求集 $Quorum_i$,生成 REQ_i 的应答集合 $Rset_i$,向 $Quorum_i$ 中的节点发送请求消息 $REQUST_i$,请求进程阻塞在请求 REQ_i 上,返回.

(3) 释放请求 **REQ_i**

(a) 删除 REQ_i,若 $RQ_i=\varnothing$,则返回.

(b) 如果 REQ_i 有对应的转移应答请求 REQ_j,则发送转移应答 $TRANS\_REP_j$,删除 REQ_j,返回;否则转下一步.

(c) 取 RQ_i 中优先权最高的 REQ_j,如果 REQ_j 是本地请求,如果 Check_R_quorum(REQ_j)=1,则唤醒阻塞在 REQ_j 上的进程,并让其进入临界区,返回;如果 Check_R_quorum(REQ_j)=0,则向 REQ_j 的应答集 $Rset_j$ 中未收到应答消息的节点重发请求 REQ_j,返回;如果 REQ_j 是外地请求,如果 REQ_j 是已应答未释放的请求,则返回,否则发送 REQ_j 的应答消息 $REPLY_j$,返回.

(4) 收到请求 **REQ_i**

(a) 如果 RQ_i 已有 REQ_i,则表示 REQ_i 是重发请求,转下一步,如果 REQ_i 比等待队列 RQ_i 中的所有请求优先级都高,则直接发送应答消息,转下一步,否则将 REQ_i 按优先级插入 RQ_i.

(b) 找出 RQ_i 中优先级最高的请求 REQ_j,如果 REQ_j 是已发送应答,但未释放的请求,则返回,否则转下一步.

(c) 如果 REQ_j 是本地请求,转*(d)*,如果 REQ_j 是外地请求,发送 REQ_j 的应答消息 $REPLY_j$.

(d) 如果 Check_R_quorum(REQ_j)=1,则唤醒阻塞在 REQ_j 上的进程,使其进入临界区,返回,否则转下一步.

(e) 如果 REQ_j 的应答集 $Rset_j$ 中未应答的节点活跃,则向 $Rset_j$ 中未收到应答消息的节点重发请求 REQ_j,返回,否则转*(f)*.

(f) 找出本节点 S_i 的第一个节点全活跃的备选请求集 $Rset_i$,按该请求集重新生成 REQ_j 应答集合,并向 $Rset_i$ 中的节点发送 REQ_j 的请求消息 $REQUST_j$,返回,如果没有节点全活跃的备选请求集,则 S_i 调用请求集重组过程生成本节点的请求集,转*(f)*.

(5) 收到应答 **$REPLY_i$**

(a) 找出 RQ_i 中 $REPLY_i$ 对应的 REQ_i,如果 $REPLY_i$ 消息携带的后续请求 REQ_j 比已收到的最高优先级后续请求的优先级高,则令 REQ_j 为最高优先级后续请求,并保存发送的节点号.

(b) 将 REQ_i 的应答集合 $Rset_i$ 置位.

(c) 如果 Check_R_quorum(REQ_i)=1,转下一步,否则转*(5)*.

(d) 如果 REQ_i 的优先级最高,则唤醒阻塞在 REQ_i 上的进程,使其进入临界区,返回,否则返回.

(e) 如果 REQ_i 的应答集 $Rset_i$ 中未应答的节点活跃,向 $Rset_i$ 中未收到应答消息的节点重发请求 REQ_i,否则转下一步.

(f) 找出本节点 S_i 的第一个节点全活跃的备选请求集 $Rset_i$,按该请求集重新生成 REQ_j 应答集合,并向 $Rset_i$ 中的节点发送 REQ_j 的请求消息 $REQUST_j$,返回,如果没有节点全活跃的备选请求集,则 S_i 调用请求集重组过程生成本节点的请求集,转*(f)*.

(6) 收到转移应答 **$TRANS\_REP_i$**

(a) 找出 $TRANS\_REP_i$ 对应的 REQ_i,并将其 $Rset_i$ 置位.

(b) 如果 Check_R_quorum(REQ_j) = 1, 转下一步, 否则转 *(d)*.

(c) 如果 REQ_i 的优先级最高, 则唤醒阻塞在 REQ_i 上的进程, 并使其进入临界区, 返回, 否则返回.

(d) 如果 REQ_i 的应答集 $Rset_i$ 中未应答的节点活跃, $Rset_i$ 中未收到应答消息的节点重发请求 REQ_i, 否则转下一步.

(e) 找出本节点 S_i 的第一个节点全活跃的备选请求集 $Rset_i$, 按该请求集重新生成 REQ_j 应答集合, 并向 $Rset_i$ 中的节点发送 REQ_j 的请求消息 $REQUST_j$, 返回, 如果没有节点全活跃的备选请求集, 则 S_i 调用请求集重组过程生成本节点的请求集, 转*(e)*.

(7) 收到释放 **RELEASE**$_i$

(a) 删除比请求 REQ_i 优先级高的所有已发送应答的请求.

(b) 找出 RQ_i 中优先级最高的请求 REQ_j.

(c) 如果 REQ_j 是本地请求, 转*(d)*, 如果 REQ_j 是外地请求转*(f)*.

(d) 如果 Check_R_quorum(REQ_j) = 1, 则唤醒阻塞在 REQ_j 上的进程, 使其进入临界区, 返回, 否则转*(e)*.

(e) 如果 REQ_j 的应答集 $Rset_j$ 中未应答的节点活跃, $Rset_j$ 中未收到应答消息的节点重发请求 REQ_j, 否则转*(g)*.

(f) 如果 REQ_j 是已发送应答, 但未释放的请求, 则返回, 否则发送 REQ_j 的应答消息 $REPLY_j$, 返回.

(g) 找出本节点 S_i 的第一个节点全活跃的备选请求集 $Rset_i$, 按该请求集重新生成 REQ_j 应答集合, 并向 $Rset_i$ 中的节点发送 REQ_j 的请求消息 $REQUST_j$, 返回, 如果没有节点全活跃的备选请求集, 则调用请求集重组过程生成本节点的请求集, 转*(g)*.

(8) 请求集重组

如果本节点 S_i 检测到节点失效并发现所有请求集不可用, 则删除所有失效节点的请求, 并调用请求集生成算法重新生成本节点及所有节点的请求集, 并更新所有请求的请求集及应答集合.

转移释放消息代替了后续请求对应应答节点的应答消息, 使同步时间降到了 T.

6　性能分析

分布式互斥算法的性能度量主要在三个指标, 消息复杂度, 同步时间和容错能力. 本文也将分析所提出算法的这三个指标.

6.1　消息复杂度

分布式互斥算法的消息复杂度与互斥请求集的大小及完成互斥的算法有关. 设互斥请求集大小为 m, 一般算法的每次请求所需消息数为 km, k 为常数. 例如 Maekawa 算法每次请求消息数为 $3m-5m$. 基于逻辑拓扑结构的算法请求集能够通过观察得到, 其每次请求的消息数与算法的消息复杂度都很容易计算. 由于本文算法是基于实现提出的, 所以其请求集大小与消息复杂度都不能通过观察获取, 它们需通过计算机求得. 表 1 是几种算法的请求集大小对比.

表 **1**　几种典型算法的请求集长度的比较

N	7	13	21	31	43	57	73	91	111	133	157	183
$\sqrt{N}$	3	4	5	6	7	8	9	10	11	12	13	14
$N^{0.63}$	4	6	7	9	11	13	15	18	20	22	25	27
$2\sqrt{N}$	6	8	10	12	14	16	19	20	22	24	26	28
$\sqrt{2N}$	4	5	6	8	9	11	12	13	15	16	18	19
cyclic	3	5	6	7	9	9	13	14	15	19	20	22

N	211	241	273	307	343	381	421	463	507	700	1000	1200
$\sqrt{N}$	15	16	17	18	19	20	21	22	23	27	33	36
$N^{0.63}$	30	32	35	28	40	43	46	48	51	63	78	88
$2\sqrt{N}$	30	32	34	36	38	40	42	44	46	53	64	70
$\sqrt{2N}$	21	22	23	25	26	28	29	30	32	37	45	49
cyclic	24	26	28	31	32	35	37	40	41	50	63	68

从表 1 看到, 当 N 较小时*(*$N<73$*)*, 本文算法产生的请求集与 $N^{0.63}$, $\sqrt{2N}$ 算法基本相同, $2\sqrt{N}$ 较小. 当 N 较大时, 本文算法产生的请求集大小有逐渐趋向于 $2\sqrt{N}$ 算法的趋势. 又由于本文算法在处理请求集中节点失效时, 要求重新寻找节点全活跃的请求集, 并重新发送请求. 因此, 本文算法在无节点失效情况下的一次请求所需消息数为 $3m$. 在一次重新发送请求情况下的一次请求所需消息数为 $6m$, 多节点失效情形可以类推. 因此在极限情况下, 本算法的消息复杂度为 $O(2\sqrt{N})$. 虽然它较 $\sqrt{2N}$ 算法大, 但其对称性较 $\sqrt{2N}$ 与 $2\sqrt{N}$ 都好. 因为 $\sqrt{2N}$ 算法是基于三角形结构的, 在节点数不能组成完整的直角三角形时, 其节点请求集产生是不对称的. $2\sqrt{N}$ 是基于正方形的, 当节点数不是平方数时, 也不能产生完全对称的请求集. 因此, 本算法虽然在极限情况下的消息复杂度接近 $O(2\sqrt{N})$, 但其对称性能较 $\sqrt{2N}$ 与 $2\sqrt{N}$ 都有较大改善.

6.2　同步时间

本算法由于采用了转移释放消息, 因此将系统中的请求组成了一个类似于 RA 算法的动态逻辑令牌环, 使系统中的请求能够按 Lamport 时戳决定的优先级顺序依次完成. 同时, 转移释放消息改变了经典 Maekawa 算法要求一个请求释放以后才能对次优请求发送应答的规定, 次优请求发起节点收到转移释放消息后能够将其当成请求控制节点*(*两个请求集的交集*)*发送的应答消息. 因此, 次优请求发起节点进入临界区只须等待一次消息*(*转移释放消息*)*传递的时间, 即同步时间为 T. 而经典 Maekawa 算法的次优请求发起节点进入临界区必须等待两次消息*(*释放, 应答*)*传送时间, 因此其同步时间为 $2T$.

6.3　容错性能

分布式容错分节点容错与通信容错.

本算法采取了对未收到应答节点重发请求的措施保证在偶然通信故障时节点间互斥的顺利进行. 由长期通信故障造成的请求或应答消息丢失, 必须通过节点容错处理. 重发请求不仅能够实现对由偶然通信故障造成的请求或应答消息丢失, 而且能够当成对未发送应答的节点的催促消息. 对偶然通信故障造成的释放消息丢失, 通过删除比此请求优先级高的请求的方式实现容错. 因为在通信非永久性失效时不是

每一释放消息都不能到达节点,因此,任何一个异地请求总有机会被删除.这减少了请求队列中死锁的可能.同时由于采用 Lamport 时戳算法决定请求的优先级,不会有节点间循环请求资源的情况产生,避免了死锁发生的一个重要条件.

对节点失效的容错,本算法采用重新寻求请求集并重新发送请求的方式.在系统产生的请求集都不可用时,本算法采用了重新计算请求集的方式保证算法继续运行.因此,在除请求节点以外任意数量请求节点失效,本算法都能保证互斥访问的顺利完成.因此,本算法的节点容错能力能够达到 $N-1$.

7 结语

公平、健壮和易于实现的分布式互斥算法对分布式系统保证数据一致性、逻辑一致性以及时序一致性至关重要.以前的分布式互斥算法,除 Lamport 算法,RA 算法和 $N^{0.63}$算法外,都只是在节点数目与请求集大小存在一定关系时才是公平和对称的,在大多数情况下是不对称的.而且这些算法的同步时间,容错性能与消息复杂度之间存在做一定矛盾,不能三者兼顾.本文算法在节点数为任意值时都是公平和对称的,其消息复杂度较低,同步时间为 T,容错能力能够达到 $N-1$,且容易实现.因此,这是一个性能很高的分布式互斥算法.

参考文献:

[1] L Lamport. Time, clocks and ordering of events in distributed systems [J]. Comm ACM, 1978, 21(7): 558-565.

[2] G Ricart, A K Agrawala. An optimal algorithm for mutual exclusion in computer networks[J]. Communications of the ACM, 1981, 24(1): 9-17.

[3] M Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems[J]. ACM Trans Computer Systems, 1985, 3(2): 145-159.

[4] J Helary. A general scheme for token and tree-based distributed mutual exclusion algorithms[J]. IEEE Trans Parallel and Distributed Systems, 1994, 5(11): 1185-1196.

[5] M Naimi. An improvement of the Log(n) distributed algorithm for mutual exclusion[A]. Proc. Seventh Int'l Conf. Distributed Computing System[C]. IEEE, 1987. 371-375.

[6] K Raymond. Tree-based algorithm for distributed mutual exclusion[J]. ACM Trans Computing Systems, 1989, (2): 61-77.

[7] A Kumar. Hierarchical quorum consensus: a new algorithm for managing replicated data[J]. IEEE Trans Computers, 1991, (9): 996-1004.

[8] Y C Kuo, S T Huang. A geometric approach for constructing coteries and k-coteries[J]. IEEE Trans. Parallel and Distributed Systems, 1997, 8(4): 402-411.

[9] K Ogata, K Futatsugi. Formally modeling and verifying Ricart&Agrawala distributed mutual exclusion algorithm[A]. Quality Software, 2001. Proceedings. Second Asia-Pacific Conference on[C]. IEEE, 2001. 357-366.

[10] Jiannong Cao, Jingyang Zhou. An efficient distributed mutual exclusion algorithm based on relative consensus voting[A]. Parallel and Distributed Processing Symposium[C]. IEEE, 2004. 51.

[11] R Baldoni, A Virgillito. A distributed mutual exclusion algorithm for mobile *ad-hoc* networks[A]. Computers and Communications, Proceedings. ISCC 2002. Seventh International Symposium on[C]. IEEE, 2002. 539-544.

[12] T Harada, M Yamashita. Transversal merge operation: a nondominated coterie construction method for distributed mutual exclusion[J]. Parallel and Distributed Systems, IEEE Transactions on, 2005, 2(2): 183-192.

[13] S Rangarajan. A fault-tolerant algorithm for replicated data management[J]. IEEE Trans Parallel and Distributed Systems, 1995, 6(12): 1271-1282.

[14] M Bearden. A fault-tolerant algorithm for decentralized on-line quorum adaptation[A]. Fault-Tolerant Computing, 1998. Twenty-Eighth Annual International Symposium on[C]. 1998. 262-271.

[15] C Guohong. A delay-optimal quorum-based mutual exclusion algorithm for distributed systems[J]. IEEE Trans Parallel and Distributed Systems, 2001, 12(12): 1256-1268.

[16] C Guohong. A delay-optimal quorum-based mutual exclusion scheme with fault-tolerance capability[A]. Distributed Computing Systems, Proceedings. 18th International Conference on[C]. IEEE, 1998. 444-451.

[17] Wai-Shing Luk. Two new quorum based algorithms for distributed mutual exclusion[A]. Distributed Computing Systems, 1997, Proceedings of the 17th International Conference on[C]. IEEE, 1997. 100-106.

[18] W K Ng, C V Ravishankar. Coterie templates: a new quorum construction method[A]. Distributed Computing Systems, 1995, Proceedings of the 15th International Conference[C]. IEEE, 1995. 92-99.

作者简介:

李美安 男,1973 年生,博士研究生,研究方向为分布式并行计算,宽带网络与通信. E-mail: Limeian@sohu.com.

刘心松 男,1940 年生,教授,博士生导师,研究方向为分布式计算、宽带网络与通信等.